УДК 336(075.8)
DOI: 10.25559/SITITO.019.202301.163-171

Оригинальная статья

Применение технологий обучения с подкреплением и параллельного программирования для генерации и валидации программного кода

В. Е. Марченко, П. В. Никитин*, Р. И. Горохова
ФГБОУ ВО «Финансовый университет при правительстве Российской Федерации» г. Москва, Российская Федерация
Адрес: 125993, Российская Федерация, г. Москва, Ленинградский пр., д. 49/2
* pvnikitin@fa.ru

Аннотация

На текущий момент нейросети способны создавать то, что раньше считалось недосягаемым: фотореалистичные лица людей; полноценные картины по грубым наброскам; любые изображения по краткому текстовому описанию; стихи и прозу по первым строчкам или заданной теме. Все это стало возможно благодаря стремительному прогрессу в таких областях как обработка естественного языка и машинное зрение. Нейросети способны генерировать контент на основе тех данных, которые они запомнили во время обширного процесса обучения. Задачи на логику, математику и логические рассуждения является примером гибкого интеллекта, и он требует совершенно других подходов к обучению. Представленное в статье исследование предлагает выработку методологии проектирования и обучения нейросетей направленных на создание функционирующего кода. Основой исследования является возможность применения искусственного интеллекта, в частности нейронных сетей, на генерацию кода машиной, то есть задач AI4Code. В исследовании рассмотрены положения в пользу применения обучение с подкреплением в сравнении с языковыми моделями, а также архитектура необходимой для такого обучения среды. Основной исследования является направленность на использование графических ускорителей Nvidia и использование центральных процессов различных архитектур. В статье рассмотрены особенности создания среды обучения, достоинства и недостатки платформы CUDA, проведен анализ потенциальной эффективности каждого из подходов.

Ключевые слова: обучение с подкреплением, программный код, нейронные сети, среда обучения, обучающие платформы

Конфликт интересов: авторы заявляют об отсутствии конфликта интересов.

Для цитирования: Марченко В. Е., Никитин П. В., Горохова Р. И. Применение технологий обучения с подкреплением и параллельного программирования для генерации и валидации программного кода // Современные информационные технологии и ИТ-образование. 2023. Т. 19, № 1. С. 163-171. doi: https://doi.org/10.25559/SITITO.019.202301.163-171

Original article

Application of Reinforcement Learning and Parallel Programming Technologies for Program Code Generation and Validation

V. E. Marchenko, P. V. Nikitin*, R. I. Gorokhova
Financial University under the Government of the Russian Federation, Moscow, Russian Federation
Address: 49/2 Leningradsky Prospekt, Moscow 125167, Russian Federation
* pvnikitin@fa.ru

Abstract
At the moment, neural networks are able to create what was previously considered inaccessible: photorealistic faces of people; full-fledged paintings based on rough sketches; any images with a short text description; poems and prose on the first lines or a given topic. All of this has been made possible by rapid advances in areas such as natural language processing and machine vision. Neural networks are capable of generating content based on the data they have memorized during an extensive learning process. Problems for logic, mathematics and logical reasoning are an example of flexible intelligence, and it requires completely different approaches to learning. The study presented in the article proposes the development of a methodology for designing and training neural networks aimed at creating a functioning code. The basis of the study is the possibility of using artificial intelligence, in particular neural networks, to generate code by a machine, that is, AI4Code tasks. The study examined the provisions in favor of the use of reinforcement learning in comparison with language models, as well as the architecture of the environment necessary for such learning. The main research is the focus on the use of Nvidia graphics accelerators and the use of central processes of various architectures. The article discusses the features of creating a learning environment, the advantages and disadvantages of the CUDA platform, and analyzes the potential effectiveness of each of the approaches.

Keywords: reinforcement learning, program code, neural networks, learning environment, learning platforms

Conflict of interests: The authors declare no conflict of interest.

For citation: Marchenko V.E., Nikitin P.V., Gorokhova R.I. Application of Reinforcement Learning and Parallel Programming Technologies for Program Code Generation and Validation. *Modern Information Technologies and IT-Education.* 2023;19(1):163-171. doi: https://doi.org/10.25559/SITITO.019.202301.163-171

Введение

Считается, что стремительный прогресс в машинном зрении обусловлен появлением датасета ImageNet [1]. Он предоставил исследователям стандартизированный метод оценки аккуратности распознавания и огромный массив вручную аннотированных изображений разбитых на большое количество различных классов. В прошлом году компания IBM представила датасет CodeNet [2], который включает в себя широкий набор фрагментов кода написанных на 55 языках программирования шесть из которых (C++, Python, Java, C, Ruby, C#) составляют 95% от общего объема. Помимо кода каждый пример включает в себя метаданные такие как: язык, на котором написано программа, время выполнения, объем затраченной памяти, статус предложенного решения (Выполнено/Ошибка компиляции/Ошибка исполнения/Превышено ограничение памяти и др.) рейтинг и др. Эта выборка без сомнения положительно повлияет на развитие дисциплины AI4Code поэтому проработка методологии решения таких задач актуальна как никогда.

Цель исследования

В исследовании поставлена цель изучить какие подходы для генерации кода используются в современных исследованиях, выявление их преимуществ и недостатков, а так же нахождение перспективных методов для развития.
Дополнительной целью является изучение аппаратных особенностей платформы CUDA, для оценки возможности ее использования в качестве среды обучения с подкреплением

Текущее состояние проблемы исследования

Появление в 2017 году архитектуры трансформер [3] значительно повлияло на успехи в обработке естественного языка (NLP). Одним из направлений развития этой архитектуры стало соединение нескольких трансформеров последовательно. После того как несколько моделей не отличаясь архитектурой, но отличаясь размерами, стали показывать точность столь большую, во сколько раз одна модель превосходит по количеству параметров другую, в научных кругах появилось мнение, что точность и возможность нейросети напрямую зависят от её размеров. Кульминацией такого подхода стала GPT-3 [4], она оказала значительное влияние на индустрию, генерируя текст неотличимый от написанного человеком. При этом она в 100 раз превосходит по размеру своего предшественника GPT-2 [5] и в 1000 раз своего прародителя GPT[1]. Проблема такого рода нейросетей заключается в том, что они требуют огромных ресурсов для обучения. GPT-3 была обучена на 0.5×10^9 токенах текста и требовала 3×10^{23} FLOP для обучения[2], что эквивалентно 22 000 GPU дней на одной из самых мощных (на текущий момент) видеокарт Nvidia A100. Такой объём вычислений доступен только очень малому количеству исследователей, ввиду чего только единичные компании могут позволить себе разрабатывать такие модели. Несмотря на это, существуют проекты, расширяющие возможности GPT-3 [6-9].
Codex [8] модификация GPT-3 дополнительно обучена на 160 ГБ кода на языке Python. Это позволило ей стать ядром системы GitHub Copilot – технологии, позволяющей программистам по короткому текстовому описанию получать если не полностью функциональный, то, как минимум корректно структурированный код. Также кодекс может создавать документацию, описывающую функционал кода, добавлять комментарии, модифицировать уже существующий код. В исследовании [10] нейросеть GPT-3 была дообучена на небольшом наборе уравнений из различных курсов высшей математики. После чего она смогла давать текстовые решения задач, а также математические формулы и графики.
AlphaCode [11] от компании DeepMind тоже основан на архитектуре трансформер, но его главной целью является решение задач олимпиадного программирования на основании их текстового описания. Эта модель в 90% случаев генерирует рабочие решения, но огромная часть её предположений отсеивается проверкой примера, приведённого в условии задачи.
Большинство существующих проектов основаны на трансформерах и по большей части опираются на языковой аспект проблемы, ввиду чего результативность технической части остается не высокой. Такой подход хоть и показывает многообещающие результаты, не подходит для генерации функционирующего кода, но отлично подходит для резюмированная и автозаполнения кода. Поэтому трансформеры должны оставаться частью такого рода моделей, но отвечать они должны за взаимодействие с человеком, а за взаимодействие с оборудованием должна отвечать другая часть, созданная на другой архитектуре.

Обучение с подкреплением для решения задачи написания программного кода

Сейчас многие компании ведут исследования в области обучения с подкреплением, к примеру OpenAI [12], NVIDIA [13] и DeepMind [14]. Их работы показывают какими эффективными могут быть такие методы обучения. Эти исследования направлены на создание подвижного интеллекта. Такой вид интеллекта создан для решения нестандартных задач, а также нахождения своих методов достижения цели, не заложенных исследователями. И именно такой подход является наиболее подходящим для задач AI4Code.
Структура задачи написания кода аналогична игре. Каждая инструкция эквивалентна возможному ходу, который может предпринять интеллектуальный агент. Его конечная цель, будь то создание рабочей программы или победа над соперником могут быть достигнуты совершенно разными путями, поэтому нахождение правильной комбинации ходов и анализ собственного состояния – ключ к достижению цели. AlphaGoZero [15] показала, что для достижения высокого уровня в логической игре даже не нужно участие человека. Анализ собственных ходов и нахождение лучшей комбинации их позволили AlphaZero превзойти AlphaGo [16] это значит, что будущий искусственный интеллект может превзойти достижения человечества, ввиду отсутствия

[1] Radford A., Narasimhan K., Salimans T., Sutskever I. Improving Language Understanding with Unsupervised Learning. Technical Report, OpenAI, 2018.

[2] AI and compute : официальный сайт [Электронный ресурс]. URL: https://openai.com/research/ai-and-compute (дата обращения: 11.02.2023).

тех предубеждений, которые сложились во время изучения этих проблем человеком. Такое возможно благодаря тому, что агент остается со своей проблемой один на один. Человек не предоставляет примеры решений, он лишь оценивает то насколько близко модель подошла к решению проблемы. По этой причине сложность обучения достаточно высока. Исследователи должны сперва создать среду, в которой будет находиться агент, затем придумать, как действия агента будут оцениваться. Если вознаграждение достаточно отдалено от начального состояния, то большой промежуток времени действия агента будут носить хаотический характер. Из-за этого требуется большой объем симуляций среды. По этой причине логичным решением будет применение того вида вычислительных машин, которые предоставляют максимальную параллельность. Одним из этих решений могут быть видеокарты. Они состоят из большого количества независимых вычислительных блоков каждый из которых может запускать различные фрагменты кода, примененные сразу к массиву значений. Это как раз то, что подходит для решения нашей проблемы. Запуская на каждом вычислительном блоке программу, сгенерированную агентом и исполняя её на множестве начальных значений, мы за очень короткий промежуток времени можем получить большой объем проверенных вариантов, причём оценка каждого из них может быть посчитана по количеству тех начальных условий, которые привели к правильному решению. При этом множество оценок мы получаем при выполнении кода всего лишь единожды, так как все эти проверки происходит параллельно внутри одного ядра. Дополнительным плюсом будет то, что такой подход можно линейно масштабировать увеличением количества видеокарт. И так как доступность этих вычислительных ресурсов сейчас высока как никогда, то это направление исследований стоит выбрать как приоритетное.

Архитектура среды обучения на платформе CUDA

Nvidia как крупный разработчик и производитель графических карт доминирует в сегменте пользовательских, рабочих и северных ускорителей. И если превосходство в сегменте потребительской электроники начинает уменьшаться, то на рынке северных и суперкомпьютерных решений он является безоговорочным лидером последнее десятилетие. Это подтверждается тем, что каждый провайдер облачных вычислений для машинного обучения обладает, как минимум одним видом видеокарт NVIDIA, а чаще всего имеет в распоряжении только видеокарты NVIDIA различных моделей. Облачные провайдеры, на текущий момент, обладают большим набором различных архитектур и решений. Нивелировать данную особенность позволяет обширная документация доступная для разработчиков. Дополнительным плюсом является то, что цена часа работы графического ускорителя, хоть и зависит от поколения видеокарт, но даже в худшем случае, остается в рамках 1-2 долларов за час для самых новых моделей и 0.1-0.5 доллара за час для карт 4-6 летней давности[3]. Поэтому количество и масштабы исследований продолжают расти.

Ввиду всех вышеперечисленных факторов в этой статье рассмотрена возможная архитектура среды обучения на платформе CUDA. Преимуществом этой платформы является поддержка разработчиков производителем и наличие большого объема исследований на эту тему.

Для начала стоит разобраться, что из себя представляет видеоускоритель. Видеокарта, как и компьютер состоит из процессора (в данном случае графического) и памяти, но они находятся на отдельной палате. Взаимодействие между двумя процессами происходит посредством линий PCI-Express, за счёт чего возможно подключения нескольких видеокарт к одному центральному процессору. В терминологии NVIDIA они называются девайсы и хост соответственно. В графическом процессоре помимо блоков, отвечающих за обработку и отрисовку графики, присутствуют структуры, отвечающие за выполнение универсальных операций. На кристалле их несколько и называются они SM (Streaming Multiprocessor).

Р и с. 1. Структура SM блока архитектуры Volta

F i g. 1. Structure of the SM block of the Volta architecture

Источник: здесь и далее в статье все рисунки составлены авторами.

Source: Hereinafter in this article all figures were made by the authors.

Каждый SM разделен на 4 блока (Warp) в каждом из которых есть свой планировщик (обозначен оранжевым) и диспетчер инструкций (обозначен коричневым). Каждый такой блок имеет собственный набор регистров (обозначены синим) и ядер (обозначены оттенками зеленого) для вычислений операций с плавающей точкой одинарной (FP32) и двойной точ-

[3] GPU pricing // Google Cloud, 2023[Электронный ресурс]. URL: https://cloud.google.com/compute/gpus-pricing (дата обращения: 11.02.2023).

ности (FP64), целочисленных операций (INT) и добавленные в поколении Volta ядра матричных операций (Tensor core). Все эти ядра обслуживают блоки загрузки и сохранения данных (темно-красный), они связаны сразу с несколькими уровнями памяти (голубой). Объем и время доступа зависят от того может ли к этой памяти обратиться только один поток (регистры), все 32 потока внутри одного Warp (local memory), либо все Warp (shared memory), либо все SM (Global memory). Чем больше элементов могут обратиться к памяти, тем больше она по размеру, и тем дольше обрабатываются запросы к ней. Поэтому для достижения максимальной эффективности нужно правильно использовать доступные компоненты. На рисунке 1 изображена его схема.

В документации CUDA изложены методы достижения этой цели[4]. Помимо общих подходов, существуют исследования, изучающие микроархитектуру, где в глубоких подробностях описаны не задокументированные аспекты работы конкретных поколений, видов ядер и памяти, а также соотношение между C++ кодом, инструкциями PTX[5] и процессорными командами SASS[6] [17-22].

Оценка эффективности применения различных видеокарт

В Таблице №1Б представлены характеристики видеокарт, сохраняющиеся на протяжении всех версий Compute Capability (CC) 7.0-9.0. Эти поколения были выбраны ввиду того, что Volta стала первой архитектурой имеющий независимый планировщик потоков (Independent Thread Scheduler). Он положительно сказывается на производительности в тех задачах, которые обсуждаются в нашем исследовании. В Таблице №1А представлены те характеристики, которые индивидуально зависят от поколения и модели кристалла. Здесь представлены флагманские видеокарты внутри каждого поколения. Каждая целая версия x.0 CC представляет из себя решение линейки Tesla использующиеся исключительно в датацентрах. Версии с десятичной частью 7.x 8.x – это те, которые используются в потребительских компьютерах и рабочих станциях. Часть версий CC были не учтены по причине использования таких чипов только во встроенных девайсах, которые не используются в обучении нейронных сетей.

Если грубо оценивать возможное количество параллельно запущенных программ, то видеокарта V100 сможет обрабатывать 320 независимых Warp, а H100 456. Справедливым будет замечание, что видеокарта 4090 может предложить большее количество SM блоков, а значит и большее количество независимых потоков. Но преимуществам H100 будет являться наличие DPX инструкции, которые по заявлениям производителя ускоряют алгоритмы динамического программирования, что может быть полезным для решения нашего типа задач. К сожалению, на текущий момент установить истинную эффективность нельзя, так как обе видеокарты появились на рынке только в третьем квартале 2022 года, поэтому специализированные тесты и исследования архитектур ещё не проводились.

Таблица 1А и 1Б. **Различные характеристики архитектур видеокарт NVIDIA**
Table 1A and 1Б. **Various characteristics of NVIDIA video card architectures**

Compute Capability	7.0	7.5	8.0	8.6	8.9	9.0
Поколение	Volta	Turing	Ampere	Ada Lovelance	Hopper	
Название чипа	GV100	TU102	GA100	GA102	AD102	GH100
Модели видеокарт	V100	2080Ti	A100	3090Ti	4090	H100
Количество SM в чипе	80	72	108	84	128	114
Количество ядер FP32	64	128				
Аппаратно ускоренное копирование памяти	нет	есть				
DPX инструкции	нет	есть				
Распределенная общая память	нет	есть				
Аппаратно ускоренный барьер потоков	нет	есть				
Максимальное количество активных Warp в SM	64	32	64	48	64	
Объем общей памяти в SM	96 KB	64 KB	164KB	100 KB	228 KB	

Количество потоков в Warp	32
Количество 32 битных регистров	65536
Количество одновременно исполняемых Warp	4
Максимальное количество регистров на поток	255
Количество целочисленных операций за такт	64

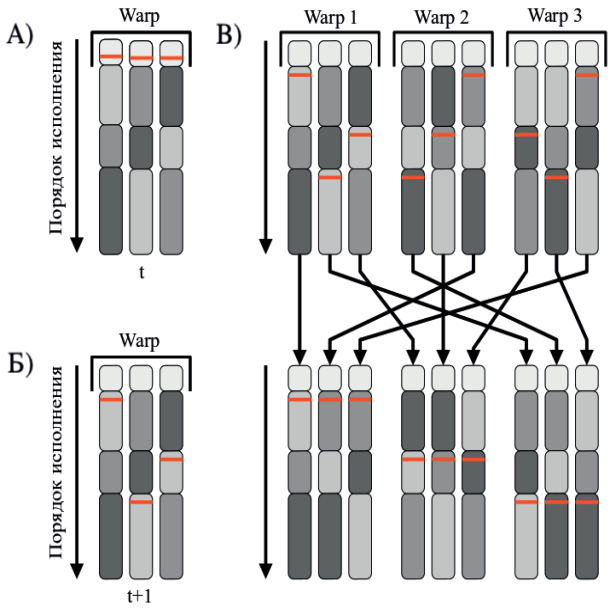
Источник: составлено авторами.
Source: Compiled by the authors.

[4] CUDA C++ programming guide [Электронный ресурс] // NVIDIA, 2023. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (дата обращения: 11.02.2023).

[5] Parallel Thread Execution ISA Version 8.3 [Электронный ресурс] // NVIDIA, 2023. URL: https://docs.nvidia.com/cuda/parallel-thread-execution/index.html (дата обращения: 11.02.2023).

[6] CUDA Binary Utilities [Электронный ресурс] // NVIDIA, 2023. URL: https://docs.nvidia.com/cuda/cuda-binary-utilities/index.html (дата обращения: 11.02.2023); Jia Z. et al. Dissecting the NVIDIA volta GPU architecture via microbenchmarking. Technical Report. Citadel Enterprise Americas LLC, 2018. URL: https://arxiv.org/pdf/1804.06826.pdf (дата обращения: 11.02.2023).

Углубляясь в детали, можно сказать, что на самом деле, каждый Warp Scheduler исполняет инструкции четырёх Warp одновременно, так как задержка исполнения инструкций составляет от четырёх тактов, то Warp Scheduler в ожидании выполнения инструкции может послать на обработку инструкции другого Warp, что скрывает задержки выполнения и повышает пропускную способность блока. Ближайшей аналогией такой технологии будет являться Hyper Threading. Она создана для решения аналогичной проблемы в центральных процессах. И делается это похожим способом: планировщик создает два логических ядра, которые для системы отображаются как два независимых ядра за счёт чего на вход одному физическому ядру поступают два потока инструкций, и пока один поток выполняется, а физическое ядро простаивает, инструкции второго логического потока исполняются им. Таким образом, получается, что одна видеокарта способна параллельно обрабатывать порядка 1,5-2 тыс Warp, что на порядок превосходит максимальное количество потоков в 64 ядерном x86 процессоре AMD Ryzen Threadripper 3990X и его серверном аналоге AMD EPYC 7773X. Помимо самого количества потоков, основным преимуществам видеокарты является тот факт что каждая инструкция исполняется на множестве ядер, ввиду чего мы имеем векторные операции. То есть одно преобразование может быть выполнено сразу на нескольких значениях. Из этого следует, что одна видеокарта может одновременно вычислить от 41 до 66 тысяч различных значений. В общем случае такая техника подсчёта корректна, но точное количество зависит как от типа данных, так и от поколения видеокарты. Учитывая, что количество ядер для операций над числами с плавающей точкой одинарной точности стало соответствовать суммарному количеству потоков в четырёх Warp только в видеокартах начиная с Compute Capability 8.6 за один такт только половина потоков в Warp могли быть выполнены. Аналогичная ситуация сохраняется на протяжении всех поколений относительно целочисленных операций (Таблица №1Б). Поэтому для задач запуска произвольного кода существуют ограничения. И пропускная способность от этого может подать до двух раз. Но частично решить эту проблему можно с помощью тензорных ядер. Они как было выяснено в исследовании [23] способны проводить до 256 операций объединённого сложения и умножения за один такт в одном warp. Однако доступно численные тензорные операции только видеокартам поколения Amper и выше и выбор типов данных ограничен (умножение с числами 8/4 бит, сложение 32 бита). У центральных процессоров также существуют специальные операции. Но самые большие из них это AVX-512 инструкции. Объем памяти, с которым они могут работать за один такт ограничивается 512 битами что аналогично объёму, который способны обработать целочисленные CUDA ядра за один такт. Это в определённых аспектах уменьшает разрыв в производительности видеокарты и процессора. Принимая во внимание все особенности как видеокарт, так и процессов, можно сказать, что хоть и потенциальный объем вычислений видеокарты в сотни раз превосходит оный у процессоров, действительная производительность может остается в пределах 1-2 порядков и при условии, что упор сделан на максимальную оптимизацию кода под параллельное выполнение.



Р и с. 2. Блок схема перегруппировки потоков в Warp
F i g. 2. Block diagram of thread regrouping in Warp

Возможные методы оптимизации разработки программного кода

Основная проблема, с которой можно столкнуться во время запуска кода на графическом процессоре это его неполное использование. Это происходит из-за ветвлений в коде. Каждый раз, когда появляется выбор между двумя путями исполнения инструкций и пути потоков внутри одного warp расходится, планировщику приходится обрабатывать каждый из путей по поддельности, отключая на это время другие потоки. Из-за этого код, который мог выполняться параллельно, исполняется последовательно и все другие потоки простаивают, снижая время использования процессорных ядер. Существует множество решений данного вида проблем, начиная с аппаратных механизмов заканчивая программными оптимизациями. К примеру, планировщик центрального процессора начинает исполнение ветки, не дожидаясь инструкции, которая переведет указатель на дальнейший код, тем самым при правильном предсказании выигрывается время, но существует риск того, что выполненный код окажется не нужным из-за того, что исполнение пошло по другому пути и тогда выигрыш в скорости пропадает. В SIMD архитектурах присутствует дополнительный подход, основанный на предикативной логике. В этом случае, вместо прыжков используются специальные регистры, в которых сохраняется нулевое значение, зависящее от выбранного пути. Каждый поток исполняет все инструкции независимо от того, какой ветке они принадлежат, но результаты операций сохраняются только в тех потоках, где эта ветка была результатом ветвления. Помимо этого, существуют так называемые, branchless программирование. В этом случае вместо if, else и switch конструкций, используются логические

и сравнительные функции, которые возвращают значение ноль или единица, тем самым при умножении сохраняют или убирают один из членов в общей сумме, которая впоследствии сохраняется в переменную. Этот подход позволяет убрать ветвление, но им нельзя выполнять последовательные операции. Все эти методы исправляют ситуацию, но лишь незначительно. Одним из более эффективных подходов может быть перераспределение потоков среди Warp в процессе исполнения. Похожий подход был реализован компанией NVIDIA в новом поколении видеокарт под названием Ada Lovelace, что говорит о его потенциальной эффективности[7]. На рисунке 2 представлена схема, описывающая данный процесс. На рисунке 2А представлен упрощенный Warp с тремя потоками разделенный на блоки последовательного исполнения. Каждый блок не содержит внутри себя ветвлений, но на границах блоков возможны условные переходы к началу других секций. Градиент в потоке указывает на то в какой последовательности блоки будут исполнены: начинаясь на более светлых сегментах и заканчиваясь более тёмными. Текущая инструкция в момент времени t обозначена красной линией. На рисунке 2Б обозначен этот же Warp в момент времени t + 1 как раз после того, как произошло ветвление. Разное положение счётчика команд указывает на то, что для дальнейшей обработки Warp блок последовательных команд каждого потока должен быть выполнен поочередно, что полностью убирает преимущество наличия 32 ядер. На рисунке 2В изображено несколько Warp. Внутри одного Warp все счётчики команд находится внутри разных блоков, среди разных Warp существуют потоки с одинаковым положением счётчика команд. Если такие потоки объединить в один Warp, то тогда исполнение снова сможет стать параллельным внутри одного Warp, но различным среди разных Warp. В случае, представленном на рисунке 2В выигрыш, может быть получен только, если время перегруппировки будет меньше, чем время исполнения двух других блоков. Иначе, если время исполнения всех трёх блоков последовательно и время перегруппировки и исполнения одного блока, одинаково или больше, то такая операция не имеет смысла. Влиять на величину временного порога, после которого имеет смысл воспользоваться перегруппировкой может как количество ветвей, так и длина блоков. Если все 32 потока должны выполнить разные инструкции, но количество этих инструкций мало, к примеру две, то если перегруппировка занимает времени больше, чем 62 инструкции то она не эффективна. Или, если число инструкция велико, порядка 16, а количество веток мало, к примеру 4, то перегруппировка продолжительностью больше 48 будет все также не эффективна. Такая перегруппировка имеет смысл только в тех случаях, когда внутри Warp потоки распределены среди большого количества веток, а сегменты последовательного кода достаточно продолжительны.

Анализ проблем развития дисциплины AI4Code

Ввиду обширности изучаемого вопроса, для всего множества исследований могут быть рассмотрены частные вопросы AI4Code. Одна из них, проблема контроля искусственного интеллекта, порожденная способностью машины находить собственные решения. Непредвиденные действия могут являться приятным дополнением к основной цели исследования, однако чаще всего они создают трудности учёным. В исследовании[8], группа ученых формулирует суть проблемы и рассказывает о премьерах её появления в уже проведённых исследованиях. В случае задач AI4Code такой феномен значительно повышает ставки: с одной стороны, нахождение нестандартных способов компиляции или новых сверх-эффективных алгоритмов влечёт за собой общий прогресс в сфере компьютерных наук, но с другой стороны, может привести к злоупотреблению машиной программными или аппаратными особенностями компьютеров [24, 25]. Это может привести к нежелательному результату, каким может являться чрезмерное использование определёнными видами команд, инструкций или программ, преодоление исследователями установленных барьеров, использование аппаратных особенностей, ограничивающих совместимость генерируемых программ среди различных видов архитектуры и операционных систем. Все эти проблемы обязательно должны быть учтены будущими исследователями, поэтому наблюдение за результатами работы машин, должно стать основной частью их работы, так как каждое нестандартное решение может принести как огромную пользу, так и создать большие трудности.

Помимо этого, стоит вопрос развертывания модели и среды на оборудовании. Для большего удобства среда должна иметь общие интерфейсы с уже существующими программными решениями, такими как TensorFlow, Keras, PyTorch. В дополнение к этому на уровне кода GPU стоит подумать либо о переключении контекста, либо об использовании нескольких видеокарт, так как глубокое обучение стало неотъемлемой частью обучения с подкреплением, а оно само требует наличия видеокарты, которая отведена симуляцию среды.

Заключение

Обозначив основные направления развития дисциплины AI4Code, а также возможности дальнейшего развития можно с уверенностью сказать, что прогресс в этой области будет напрямую зависеть от того насколько большие вычислительные ресурсы будут доступны исследователем, а также насколько эффективные алгоритмы будут запущены на этих мощностях. В проделанной работе собрана информация о глобальных трендах развития темы исследования и о том, какие знания из смежных областей необходимы для дальнейшего развития. Помимо этого рассмотрены аппаратные особенности оборудования, на котором происходит исследование в области искусственного интеллекта и предложены способы использования этих ресурсов с целью решения задач генерации кода машиной, а также проведена количественная оценка объёма одновременных вычислений и предложены методы оптимизации с целью достижения максимальной теоретической эффективности.

[7] NVIDIA Ada GPU architecture. V2.02 [Электронный ресурс] // NVIDIA Corporation, 2023. URL: https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf (дата обращения: 11.02.2023).

[8] Krakovna V. et al. Specification gaming: the flip side of AI ingenuity [Электронный ресурс] // Google DeepMind, 21 April 2020. URL: https://www.deepmind.com/blog/specification-gaming-the-flip-side-of-ai-ingenuity (дата обращения: 11.02.2023).

References

[1] Russakovsky O., Deng J., Su H. et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*. 2015;115:211-252. https://doi.org/10.1007/s11263-015-0816-y

[2] Puri R. et al. CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks. In: Vanschoren J., Yeung S. (eds.) Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks. Vol. 1. Curran; 2021. p. 1-13. Available at: https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/a5bfc9e07964f8dddeb95fc584cd965d-Paper-round2.pdf (accessed 11.02.2023).

[3] Vaswani A., Shazeer N., Parmar N. et al. Attention Is All You Need. In: Guyon I. et al. (eds.) Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Red Hook, NY, USA: Curran Associates Inc.; 2017. Vol. 30. p. 6000-6010. Available at: https://dl.acm.org/doi/pdf/10.5555/3295222.3295349 (accessed 11.02.2023).

[4] Brown T.B., Mann B., Ryder N. et al. Language Models are Few-Shot Learners. In: Larochelle H., Ranzato M., Hadsell R., Balcan M.F., Lin H. (eds.) Advances in Neural Information Processing Systems. Red Hook, NY, USA: Curran Associates Inc.; 2020. Article no. 159. p. 1877-1901. Available at: https://papers.nips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf (accessed 11.02.2023).

[5] Radford A., Wu J., Child R. et al. Language Models are Unsupervised Multitask Learners. *OpenAI blog*. 2019;1(8):9. Available at: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf (accessed 11.02.2023).

[6] Schick T., Schütze H. Few-Shot Text Generation with Natural Language Instructions. In: Moens M.-F., Huang X., Specia L., Wen-tau Yih S. (eds.) Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Punta Cana, Dominican Republic: Association for Computational Linguistics; 2021. p. 390-402. https://doi.org/10.18653/v1/2021.emnlp-main.32

[7] Winata G.I., Madotto A., Lin Z., Liu R., Yosinski J., Fung P. Language Models are Few-shot Multilingual Learners. In: Proceedings of the 1st Workshop on Multilingual Representation Learning. Punta Cana, Dominican Republic: Association for Computational Linguistics; 2021. p. 1-15. https://doi.org/10.18653/v1/2021.mrl-1.1

[8] Chen M. et al. Evaluating Large Language Models Trained on Code. *arXiv*:2107.03374. 2021. Available at: https://arxiv.org/abs/2107.03374 (accessed 11.02.2023).

[9] Rubio C., Mella F., Martínez C., Segura A., Vidal C. Exploring Copilot Github to Automatically Solve Programming Problems in Computer Science Courses. In: 2023 42nd IEEE International Conference of the Chilean Computer Science Society (SCCC). Concepcion, Chile: IEEE Computer Society; 2023. p. 1-8. https://doi.org/10.1109/SCCC59417.2023.10315758

[10] Drori I. et al. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *PNAS*. 2022;119(32):e2123433119. https://doi.org/10.1073/pnas.2123433119

[11] Li Y. et al. Competition-level code generation with AlphaCode. *Science*. 2022;378(6624):1092-1097. https://doi.org/10.1126/science.abq1158

[12] Baker B. et al. Emergent Tool Use From Multi-Agent Autocurricula. In: International Conference on Learning Representations. Addis Ababa, Ethiopia; 2020. p. 1-28. Available at: https://openreview.net/forum?id=SkxpxJBKwS (accessed 11.02.2023).

[13] Fan L. et al. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge. In: Thirty-sixth Conference on Neural Information Processing Systems (NeurIPS 2022). Track on Datasets and Benchmarks. New Orleans; 2022. p. 1-20. Available at: https://nips.cc/virtual/2022/poster/55737 (accessed 11.02.2023).

[14] Fawzi A., Balog M., Huang A. et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*. 2022;610:47-53. https://doi.org/10.1038/s41586-022-05172-4

[15] Silver D., Schrittwieser J., Simonyan K. et al. Mastering the game of Go without human knowledge. *Nature*. 2017;550:354-359. https://doi.org/10.1038/nature24270

[16] Silver D., Huang A., Maddison C. et al. Mastering the game of Go with deep neural networks and tree search. *Nature*. 2016;529:484-489. https://doi.org/10.1038/nature16961

[17] Bhagirath, Mittal N., Kumar S. Machine Learning Computation on Multiple GPU's using CUDA and Message Passing Interface. In: 2019 2nd International Conference on Power Energy, Environment and Intelligent Control (PEEIC). Greater Noida, India: IEEE Computer Society; 2019. p. 18-22. https://doi.org/10.1109/PEEIC47157.2019.8976714

[18] Diehl P., Seshadri M., Heller T., Kaiser H. Integration of CUDA Processing within the C++ Library for Parallelism and Concurrency (HPX). In: 2018 IEEE/ACM 4th International Workshop on Extreme Scale Programming Models and Middleware (ESPM2). Dallas, TX, USA: IEEE Computer Society; 2018. p. 19-28. https://doi.org/10.1109/ESPM2.2018.00006

[19] Kerr A., Diamos G., Yalamanchili S. Modeling GPU-CPU workloads and systems. In: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU-3). Association for Computing Machinery, New York, NY, USA; 2010. p. 31-42. https://doi.org/10.1145/1735688.1735696

[20] Lustig D., Sahasrabuddhe S., Giroux O. A Formal Analysis of the NVIDIA PTX Memory Consistency Model. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19). Association for Computing Machinery, New York, NY, USA; 2019. p. 257-270. https://doi.org/10.1145/3297858.3304043

[21] Abdelkhalik H., Arafa Y., Santhi N., Badawy A. -H. A. Demystifying the Nvidia Ampere Architecture through Microbenchmarking and Instruction-level Analysis. In: 2022 IEEE High Performance Extreme Computing Conference (HPEC). Waltham, MA, USA: IEEE Computer Society; 2022. p. 1-8. https://doi.org/10.1109/HPEC55821.2022.9926299

[22] van Stigt R., Swatman S.N., Varbanescu A.-L. Isolating GPU Architectural Features Using Parallelism-Aware Microbenchmarks. In: Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering (ICPE '22). Association for Computing Machinery, New York, NY, USA; 2022. p. 77-88. https://doi.org/10.1145/3489525.3511673

[23] Sun W., Li A., Geng T., Stuijk S., Corporaal H. Dissecting Tensor Cores via Microbenchmarks: Latency, Throughput and Numeric Behaviors. *IEEE Transactions on Parallel and Distributed Systems*. 2023;34(1):246-261. https://doi.org/10.1109/TPDS.2022.3217824

[24] Christiano P. et al. Deep reinforcement learning from human preferences. In: Guyon I. et al. (eds.) Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Red Hook, NY, USA: Curran Associates Inc.; 2017. Vol. 30. p. 1-9. Available at: https://proceedings.neurips.cc/paper_files/paper/2017/hash/d5e2c0adad503c91f91df240d0cd4e49-Abstract.html (accessed 11.02.2023).

[25] Schieffer G., Peng I. Accelerating Drug Discovery in AutoDock-GPU with Tensor Cores. In: Cano J., Dikaiakos M.D., Papadopoulos G.A., Pericàs M., Sakellariou R. (eds.) Euro-Par 2023: Parallel Processing. Euro-Par 2023. Lecture Notes in Computer Science. Vol. 14100. Springer, Cham; 2023. p. 608-622. https://doi.org/10.1007/978-3-031-39698-4_41

Поступила 11.02.2023; одобрена после рецензирования 19.03.2023; принята к публикации 21.03.2023.
Submitted 11.02.2023; approved after reviewing 19.03.2023; accepted for publication 21.03.2023.

Об авторах:

Марченко Вадим Евгеньевич, студент Департамента анализа данных и машинного обучения, ФГБОУ ВО «Финансовый университет при правительстве Российской Федерации» (125993, Российская Федерация, г. Москва, Ленинградский пр., д. 49/2), **ORCID: https://orcid.org/0000-0002-1003-4592**, marchenkove@gmail.com

Никитин Петр Владимирович, доцент департамента анализа данных и машинного обучения, ФГБОУ ВО «Финансовый университет при правительстве Российской Федерации» (125993, Российская Федерация, г. Москва, Ленинградский пр., д. 49/2), кандидат педагогических наук, доцент, **ORCID: https://orcid.org/0000-0001-8866-5610**, pvnikitin@fa.ru

Горохова Римма Ивановна, доцент Департамента анализа данных и машинного обучения, ФГБОУ ВО «Финансовый университет при правительстве Российской Федерации» (125993, Российская Федерация, г. Москва, Ленинградский пр., д. 49/2), кандидат педагогических наук, доцент, **ORCID: https://orcid.org/0000-0001-7818-8013**, rigorokhova@fa.ru

Все авторы прочитали и одобрили окончательный вариант рукописи.

About the authors:

Vadim E. Marchenko, Student of the Department of Data Analysis and Machine Learning, Financial University under the Government of the Russian Federation (49/2 Leningradsky Prospekt, Moscow 125167, Russian Federation), **ORCID: https://orcid.org/0000-0002-1003-4592**, marchenkove@gmail.com

Petr V. Nikitin, Associate Professor of the Department of Data Analysis and Machine Learning, Financial University under the Government of the Russian Federation (49/2 Leningradsky Prospekt, Moscow 125167, Russian Federation), Cand. Sci. (Ped.), Associate Professor, **ORCID: https://orcid.org/0000-0001-8866-5610,** pvnikitin@fa.ru

Rimma I. Gorokhova, Associate Professor of the Department of Data Analysis and Machine Learning, Financial University under the Government of the Russian Federation (49/2 Leningradsky Prospekt, Moscow 125167, Russian Federation), Cand. Sci. (Ped.), Associate Professor, **ORCID: https://orcid.org/0000-0001-7818-8013**, rigorokhova@fa.ru

All authors have read and approved the final manuscript.